Лабораторна робота №8

Виконання операцій з плаваючою точкою та вивчення команд x87 FPU

Мета: Навчитися програмувати операції з плаваючою точкою на асемблері.

Завдання:

1. Створити у середовищі MS Visual Studio проект з ім'ям **Lab8**.
2. Написати вихідний текст програми згідно варіанту завдання. У проекті мають бути головний файл **main8.asm** та інші модулі (за необхідності).
3. У цьому проекті кожний модуль може окремо компілюватися.
4. Скомпілювати вихідний текст і отримати виконуємий файл програми.
5. Перевірити роботу програми. Налагодити програму.
6. Отримати результати – файл числових значень згідно варіанту завдання.
7. Проаналізувати та прокоментувати результати, вихідний текст та дизасембльований машинний код програми.

Теоретичні відомості

Середовище x87 FPU

Блок, який отримав назву x87 FPU (*Floating Point Unit*), виконує операції у форматах з плаваючою точкою. Спочатку це був окремий пристрій (сопроцесор 8087). Починаючи з Intel 80486 такий блок міститься усередині центрального процесора.

Для виконання у середовищі x87 FPU операцій з плаваючою точкою є декілька десятків команд, яки можна використовувати у програмах на асемблері. Для цього програмістам потрібно уявляти, хоча б загалом, з чого складається блок x87 FPU, які він має ресурси, та як їх можна використовувати. Іншими словами, програмісту надається деяке середовище – у даному випадку це x87 FPU, у якому виконуватимуться його програми. Це середовище утворюється з апаратних ресурсів, інтерфейсів доступу до них, та інших дозволів та заборон, які враховуються на певному програмному рівні.

Рис. 1. Елементи середовища x87 FPU

Регістр статусу. Цей 16-бітовий регістр зберігає інформацію про стан x87 FPU у вигляді значень бітів-прапорців:

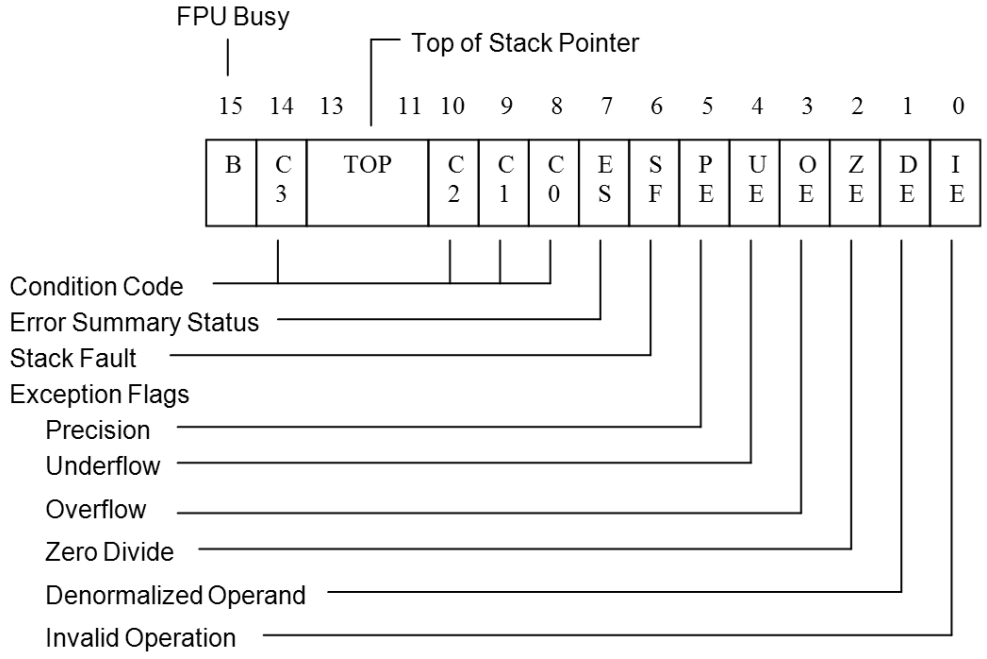

Рис. 2. Регістр статусу x87 FPU (згідно джерела: Intel® 64 and IA-32 Architectures. Software Developer's Manual).

Три біти 11-13 є значенням TOP (*Top of Stack Pointer*), яке означає адресу вершини ST(0) стеку даних. Іншими словами, TOP – це номер (від 0 до 7) регістру R_{TOP}, який на даний момент представляє вершину стеку.

Стек даних FPU

Для виконання операцій потрібно завантажувати дані у стек FPU.

Стек даних x87 FPU складається з восьми 80-бітових регістрів R0-R7. Цей стек організований як кільце. Доступ до стеку даних можливий тільки через його вершину, яка зветься ST(0). Вершині стеку відповідає один з регістрів R0-R7, на який вказує 3-х бітове значення вказівника TOP.

Запис до стеку. Це можна зробити командою FLD

FLD src

Значення операнду src записується у вершину стека ST(0). Наприклад:

```
myvar dt 35.247
. . .

fld tbyte ptr[myvar]
```

При виконанні команди FLD спочатку TOP зменшується на одиницю (причому, якщо TOP дорівнював 0, то буде TOP = 7) – так визначається нове розташування ST(0); а потім у відповідний регістр R_{TOP} записується числове значення операнду (рис. 3).

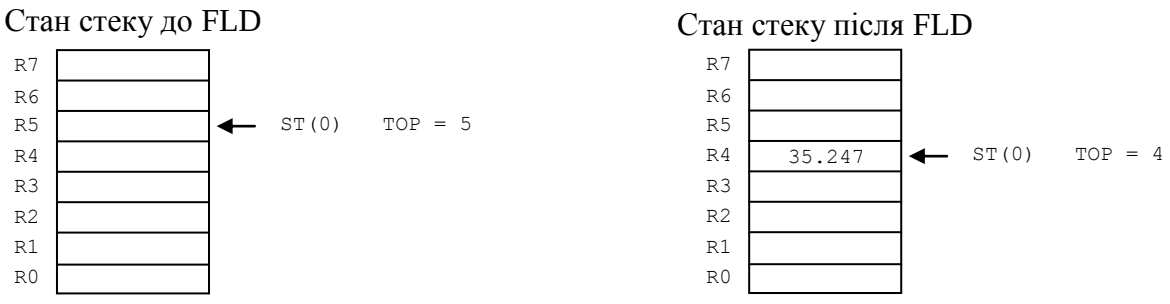

Рис. 3. Запис у стек одного числа командою FLD

Розглянемо приклад запису у стек x87 FPU декількох значень. Простежимо зміни стану стеку впродовж виконання команд FLD.

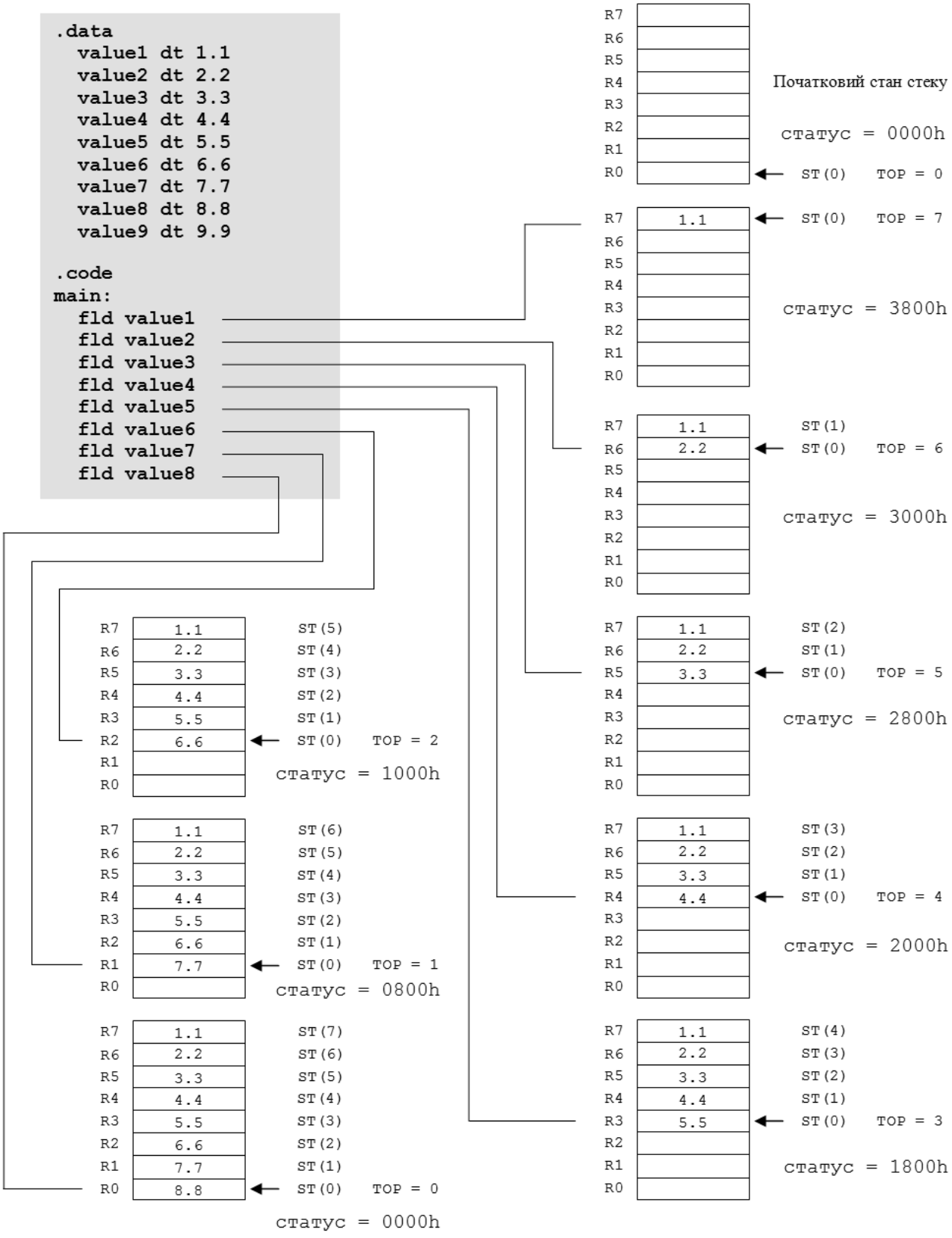

Рис. 4. Стек FPU може зберігати одночасно не більше восьми значень

Читання зі стеку. Це можна зробити командою FSTP

FSTP dest

Наприклад:

```
myvar dt ?
. . .

fstp tbyte ptr[myvar]              ; myvar = 35.247
```

Значення вершини стека ST(0) записується у операнд призначення dest. Після цього TOP збільшується на одиницю (причому, якщо TOP дорівнював 7, то буде TOP = 0) – так визначається нове розташування ST(0).

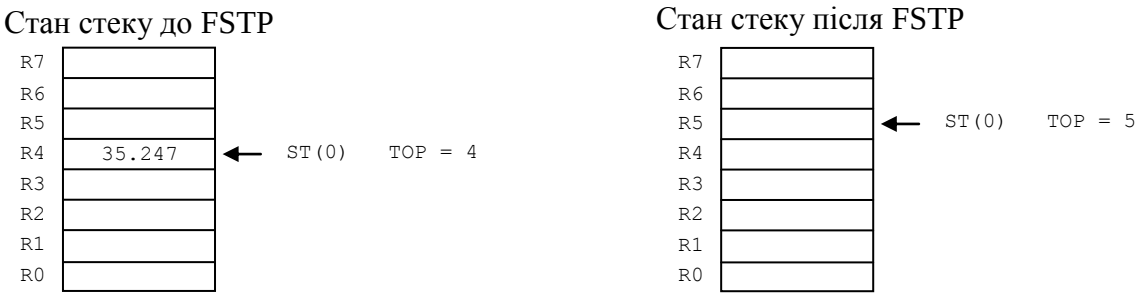

Рис. 4. Читання зі стеку одного числа командою FSTP

Операндом команди FSTP може бути 80-бітова, або 64-бітова, або 32-бітова перемінна, наприклад:

```
Var32 dd ?
Var64 dq ?
Var80 dt ?
. . .

fstp dword ptr[Var32]
fstp qword ptr[Var64]
fstp tbyte ptr[Var80]
```

При читанні значення зі стеку FPU у 32-бітову перемінну, виконується перетворення у 32-бітовий формат Single-Precision з плаваючою точкою.
При читанні значення зі стеку FPU у 64-бітову перемінну, виконується перетворення у 64-бітовий формат Double-Precision з плаваючою точкою.
Якщо такі перетворення неможливі, то у регістр статусу записується відповідне бітове значення.

Прочитати дані зі стеку також можна й командою FST

FST dest

Ця команда відрізняється від FSTP тим, що не виштовхує значення зі стеку (pop), тобто не змінює вказівник вершини стеку ST(0). Ще одна відмінність: команда FST не може прочитати 80-бітове значення – можна тільки у 64- або 32-бітовому форматах.

Короткий огляд команд x87 FPU

Імена усіх команд x87 FPU починаються з букви ‘F’. Виділимо, у першу чергу ті команди, які будуть корисні для виконання завдань лабораторної роботи.

FADD/FADDP	Додавання
FSUB/FSUBP	Віднімання
FMUL/FMULP	Множення
FDIV/FDIVP	Ділення
FABS	Абсолютна величина
FSQRT	Квадратний корінь
FSIN	Синус
FCOS	Косинус
FSINCOS	Синус та косинус
FPTAN	Тангенс
FPATAN	Арктангенс
FYL2X	Логарифм ($y*\log_2 x$)
FYL2XP1	Логарифм ($y*\log_2 (x+1)$)
F2XM1	Експонента ($2^x - 1$)
FSCALE	Множення операнду на степінь 2

Докладні відомості про команди x87 FPU можна знайти у літературному джерелі від розробників процесорів: "Intel® 64 and IA-32 Architectures. Software Developer’s Manual".

Приклад обчислень

Розглянемо приклад обчислення у середовищі x87 FPU двох пар добутків Res = A×B + C×D. Процес обчислень проілюстровано на рис. 5.

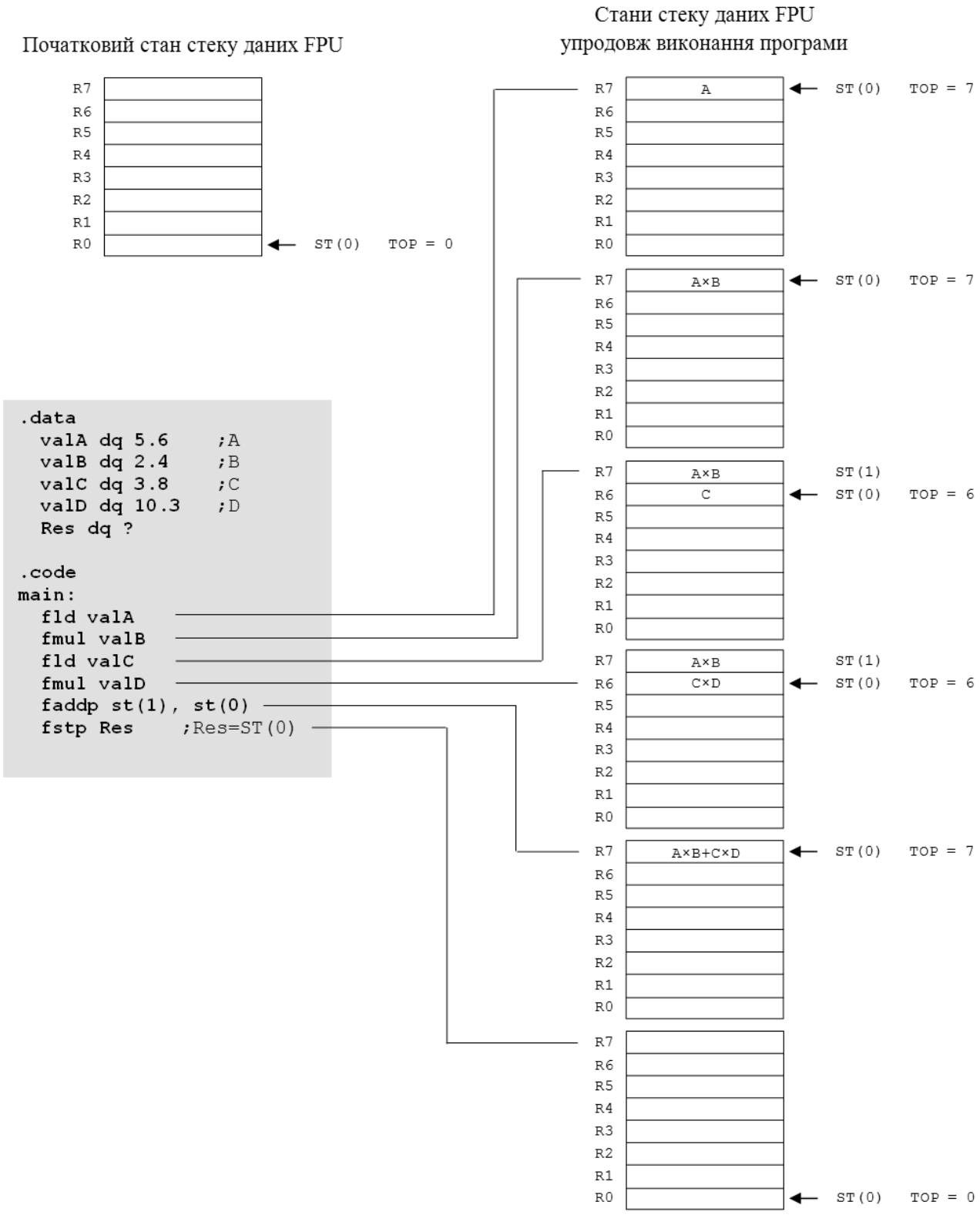

Рис. 5. Обчислення Res = A×B + C×D

Показ значень чисел, записаних у форматі з плаваючою точкою

Для того, щоб відображати результати обчислень, потрібно створити процедуру, яка перетворює значення, записане у форматі з плаваючою точкою, у текст дробу з десяткових цифр.

У попередній лабораторній роботі №3 вже були розглянуті основні відомості щодо стандартних (IEEE754) двійкових форматів з плаваючою точкою. Тепер розглянемо деякі аспекти щодо переводу у десяткову систему числення.

Звичайне число з плаваючою точкою можна описати формулою

$V = (-1)^S 2^E M,$

де:

S – знак (0 означає плюс, 1 означає мінус);

E – експонента;

M – мантиса, яка дорівнює 1.F, де F – дробова частина мантиси

Приклад двійкового коду числа у 32-бітовому форматі з плаваючою точкою

$V = 420F12A1_{16} = 01000001000001111000100101010000 1_2$

Виділимо у цього коду групи бітів елементів двійкового формату

S	e	F
0	10000100	00011110001001010100001
1	8	23

де:

S – біт знаку;

e – 8-бітовий зміщений код експоненти, причому e = E+127;

F – 23-бітова дробова частина мантиси.

Мантиса дорівнює 1.F = 1.00011110001001010100001

Обчислимо експоненту E. Оскільки у форматі записується її зміщений код

e = E + 127,

тоді

E = e – 127

Маємо $e = 10000100_2 = 132_{10}$, відповідно E = 132 – 127 = 5

Тоді число можна записати у наступному вигляді:

$V = 2^5 \cdot 1.00011110001001010100001$

Запишемо це число без експоненти, як звичайне дробове число. Для цього помножимо мантису на 2^5, тобто, виконаємо зсув мантиси вліво на 5 бітів

V = 100011.110001001010100001

Як перевести дробове двійкове число у десяткову систему? Таке завдання можна розчленувати на два кроки: окремо переводити у десятковий код цілу частину і окремо – дробову частину.

1. Перевести двійковий код цілої частини у десятковий можна послідовним **діленням на десять** у двійковому коді (двійковий код десяти є 1010_2):

$100011_2 : 1010_2 = 11_2$, залишок 101_2 – це десяткова цифра 5
$11_2 : 1010_2 = 0$, залишок 11_2 – це десяткова цифра 3

Таким чином, ціла частина у десятковому коді дорівнює 35.

2. Переведення у десятковий код дробової частини виконується **множенням на десять** у двійковому коді. Після кожного множення десяткова цифра є цілою частиною добутку. Можна порекомендувати швидкий алгоритм множення числа на 10 у двійковому коді: число (D) зсувається на біт вліво (2D) і до нього додається те саме число, зсунуте на три біти вліво (8D), тобто 10D = 2D+8D. Виконаємо це для конкретного двійкового коду дробової частини:

```
                    .110001001010100001    D1
                   1.10001001010100001    2D1
                 110.001001010100001      8D1
Перша цифра:  7 ← [111].101011101001001 01  10D1
-------------------------------------------------
                    .101011101001001 01    D2
                   1.01011101001001 01    2D2
                 101.11101001001 01       8D2
Друга цифра:  6 ← [110].1101000110111001    10D2
-------------------------------------------------
                    .1101000110111001      D3
                   1.101000110111001      2D3
                 110.1000110111001        8D3
Третя цифра:  8 ← [1000].001100010011101    10D3
-------------------------------------------------
```

```
                    .001100010011101        D4
                   0.01100010011101         2D4
                  001.100010011101          8D4
Четверта цифра: 1 ◄— [001].11101100010001   10D4
-------------------------------------------------
                    .11101100010001         D5
                   1.1101100010001          2D5
                  111.01100010001           8D5
П'ята цифра: 9 ◄— [1001].0011101010101      10D5
-------------------------------------------------
                    .0011101010101          D6
                   0.011101010101           2D6
                  001.1101010101            8D6
Шоста цифра: 2 ◄— [010].010010101001        10D6
-------------------------------------------------
```

Якщо обмежитися шістьма цифрами дробової частини, то отримаємо такий результат:

$V \approx 35.768192$

Особливі випадки у форматах з плаваючою точкою. Передбачено, що деякі комбінації бітів коду двійкового формату із плаваючою точкою використовуються для позначення особливих випадків – не число (*Not a Number – NaN*), безкінечність, нуль тощо. Ці особливі випадки позначаються кодами e = 00…0, а також e = 11…1.

Зміщена експонента e = E + зсув	Дроб F	Значення числа V	Назва числа
$e = e_{min} - 1 = 00\ldots0$	$F = 0$	$V = (-1)^S\ 0$	нуль
	$F \neq 0$	$V = (-1)^S\ 2^{Emin}\ (0.F)$	ненормалізоване число
$e_{min} \leq e \leq e_{max}$	F	$V = (-1)^S\ 2^E\ (1.F)$	нормалізоване число
$e = e_{max} + 1 = 11\ldots1$	$F = 0$	$V = (-1)^S\ \infty$	безкінечність зі знаком
	$F \neq 0$	V = NaN	не число

Алгоритм дешифрування двійкового формату з плаваючою точкою

Для перетворення у десятковий запис двійкового коду 32-бітового формату з плаваючою точкою можна рекомендувати алгоритм дешифрування (аналізу), наведений на рис. 6. Подібний алгоритм можна використати також для дешифрування 64-бітового та 80-бітового форматів.

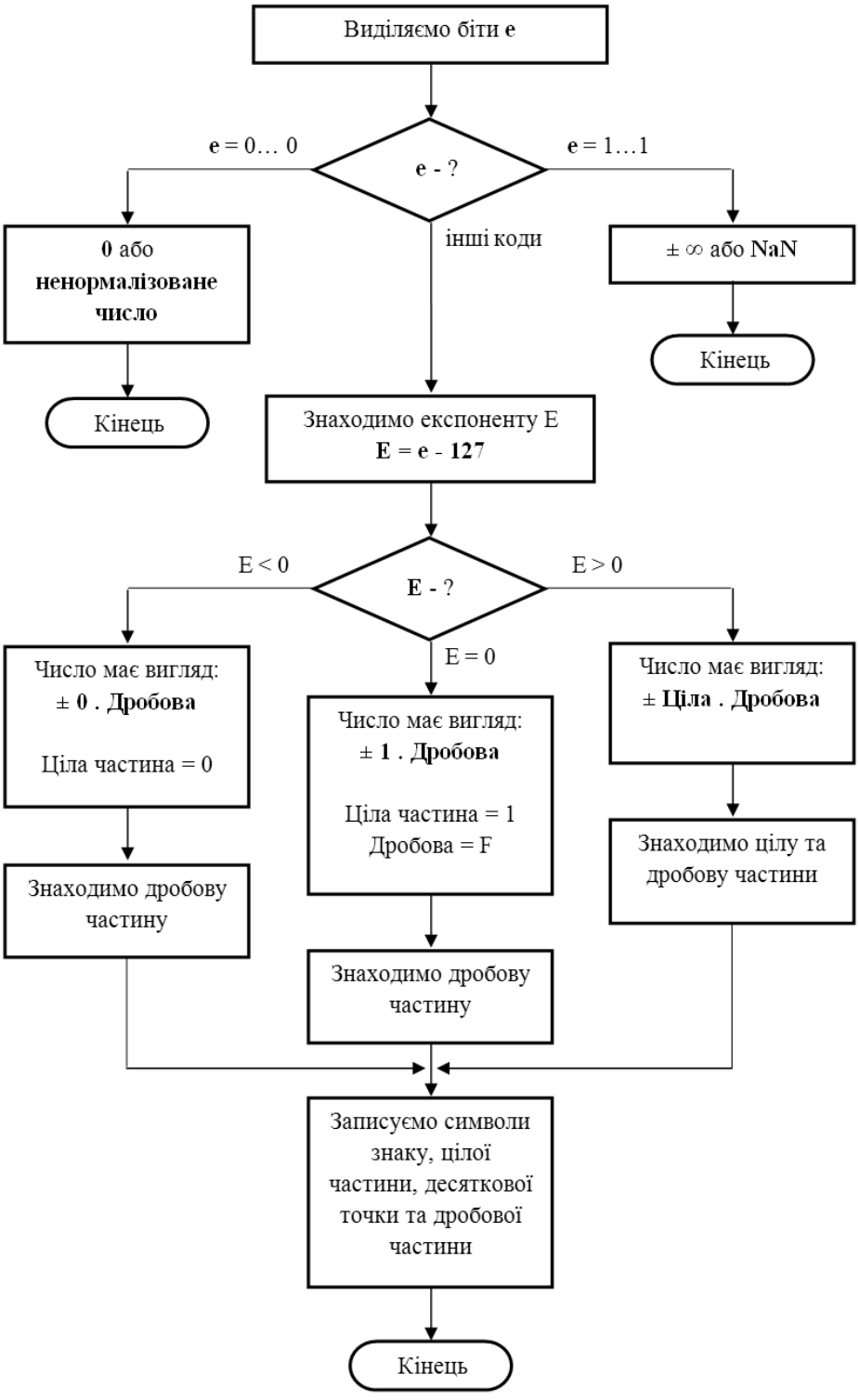

Рис. 6. Алгоритм дешифрування 32-бітового формату з плаваючою точкою

Варіанти завдання

Запрограмувати на асемблері обчислення на основі команд x87 FPU. Результат надати у вікні MessageBox у вигляді десяткового дробу з цілої частини, точки та 6 розрядів дробової частини, отриманих з 32-бітового формату з плаваючою точкою. Перетворення з двійкового формату з плаваючою точкою у рядок десяткових цифр оформити у вигляді процедури з ім'ям **FloatToDec**. Заохочення: студентам, які запрограмують таку процедуру, яка виконує перетворення у текст десяткового дробу з 64-бітового або 80-бітового форматів з плаваючою точкою, може бути суттєво підвищена оцінка.

Варіант згідно номеру студента у журналі.

№ вар	Що потрібно обчислити	Формат даних для A,B,X
1	$Res = A_0 + A_1X + A_2X^2 + \ldots + A_{n-1}X^{n-1}$	32-бітовий
2	$Res = \lvert A_0B_0 + A_1B_1 + \ldots + A_{n-1}B_{n-1} \rvert$	64-бітовий
3	$Res = A_0 + A_1 \sin B + A_2 \sin(2B) + \ldots + A_n \sin(nB)$	32-бітовий
4	X_1, X_2 = Рішення системи лінійних рівнянь 2-го порядку	64-бітовий
5	Res = квадратний корінь $(B^2_0 + B^2_1 + \ldots + B^2_{n-1})$	32-бітовий
6	$Res = A_0 + A_1 \sin X + A_2 \sin^2 X + \ldots + A_n \sin^n X$	64-бітовий
7	$Res = (A_0B_0 + A_1B_1 + \ldots + A_{n-1}B_{n-1}) / (A_0 + A_1 + \ldots + A_{n-1})$	32-бітовий
8	$Res = A_1(X-B_1) + A_2(X-B_2)^2 + \ldots + A_n(X-B_n)^n$	64-бітовий
9	$Res = (\ldots(A_{n-1}X+A_{n-2})X + A_{n-3})X + \ldots A_1)X + A_0$	32-бітовий
10	$Res = B / (A_0X^2 + A_1X^2 + \ldots + A_{n-1}X^2)$	64-бітовий
11	$Res = A_0 + A_1 \cos B + A_2 \cos^2 B + \ldots + A_n \cos^n B$	32-бітовий
12	$Res = A_0 + A_1 \cos B + A_2 \cos(2B) + \ldots + A_n \cos(nB)$	64-бітовий
13	$Res = (X-A_0)^2 + (X-A_1)^2 + \ldots + (X-A_{n-1})^2$	32-бітовий
14	Res = логарифм $(A_0B_0 + A_1B_1 + \ldots + A_{n-1}B_{n-1})$	64-бітовий
15	$Res = \lvert A_0/B_0 + A_1/B_1 + \ldots + A_{n-1}/B_{n-1} \rvert$	32-бітовий
16	Res = Детермінант 3-го порядку	64-бітовий
17	$Res = A_0 + A_1 \operatorname{tg} B + A_2 \operatorname{tg}^2 B + \ldots + A_n \operatorname{tg}^n B$	32-бітовий
18	Res = максимальне значення $(A_i - B_i)^2$, для i = 0, 1, . . ., n-1	64-бітовий
19	X_1, X_2 = Корені рівняння $AX^2 + BX + C = 0$	32-бітовий
20	Res = мінімальне значення $(A_i + B_i)^2$, для i = 0, 1, . . ., n-1	64-бітовий
21	$Res = \operatorname{arctg}(A_0/B_0) + \operatorname{arctg}(A_1/B_1) + \ldots + \operatorname{arctg}(A_{n-1}/B_{n-1})$	32-бітовий

Зміст звіту:

1. Титульний лист
2. Завдання
3. Роздруківка тексту програми
4. Роздруківка результатів виконання програми
5. Аналіз, коментар результатів, вихідного тексту та дизасембльованого машинного коду
6. Висновки

Контрольні питання:

1. Що виконується у середовищі x87 FPU?
2. Як організований стек даних x87 FPU?
3. Що таке TOP?
4. Які команди забезпечують запис та читання даних у стек x87 FPU?
5. Що таке нормалізоване число?
6. Як кодується нуль, безкінечність, не-число у форматі з плаваючою точкою?
7. Як перевести дробове число з двійкового у десятковий код?